System Evolution by Migration Coordination

Suzana Andova[2], Luuk Groenewegen[1,], and Erik de Vink[2] [*]

[1] FaST Group, LIACS, Leiden University
[2] Formal Methods Group, Department of Mathematics and Computer Science
Eindhoven University of Technology

1 Introduction

Collaborations between components can be modeled in the coordination language Paradigm [3]. A collaboration solution is specified by loosely coupling component dynamics to a protocol via their roles. Not only regular, foreseen collaboration can be specified, originally unforeseen collaboration can be modeled too [4]. To explain how, we first look very briefly at Paradigm's regular coordination specification.
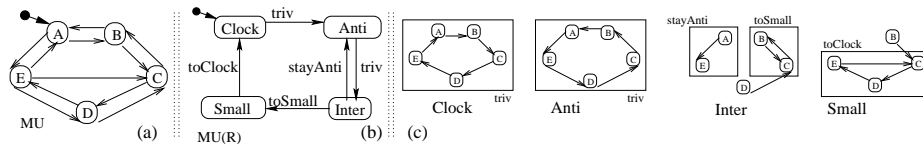

Fig. 1. Example component dynamics, role dynamics by constraints.

Component dynamics are expressed by state-transition diagrams (STDs), see Figure 1(a) for a mock-up STD *MU* in UML style. *MU* contributes to a collaboration via a role *MU(R)*. Figure 1(b) specifies *MU(R)* through a different STD, whose states are so-called phases of *MU*: temporarily valid, dynamic constraints imposed on *MU*. The figure mentions four such phases, *Clock*, *Anti*, *Inter* and *Small*. Figure 1(c) couples *MU* and *MU(R)*. It specifies each phase as part of *MU*, additionally decorated with one or more polygons grouping some states of a phase. Polygons visualize so-called traps: a trap, once entered, cannot be left as long as the phase remains the valid constraint. A trap having been entered, serves as a guard for a phase change. Therefore, traps label transitions in a role STD, cf. Figure 1(b).

Single steps from different roles, are synchronized into one protocol step. A protocol step can be coupled to one detailed step of a so-called manager component, driving the protocol. Meanwhile, local variables can be updated. It is through a consistency rule, Paradigm specifies a protocol step: (i) at the left-hand side of a $*$ the one, driving manager step is given, if relevant; (ii) the right-hand side lists the role steps being synchronized; (iii) optionally, a change clause [2] can be given updating variables, e.g. one containing the current set of consistency rules. For example, a consistency rule without change clause,

$$MU_2: A \to B \ * \ MU_1(R): Clock \overset{triv}{\to} Anti, \ MU_3(R): Inter \overset{toSmall}{\to} Small$$

where a manager step of MU_2 is coupled to the swapping of MU_1 from circling clock-wise to anti-clock-wise and swapping MU_3 from intermediate inspection into circling on a smaller scale.

2   Migration by constraint manipulation

For modeling unforeseen change, the special component *McPal* is added to a Paradigm model. *McPal* coordinates the migration towards the new way of working, by explicitly driving an unforeseen protocol. During the original, stable collaboration stage of the running Paradigm model, *McPal* is stand-by only, not influencing the rest of the model at all. This is *McPal*'s hibernated form. But, by being there, *McPal* provides the means for preparing the migration as well as for guiding the execution accordingly. To that aim, connections between *McPal* and the rest of the model are in place, realizing rudimentary interfacing for later purposes: in Paradigm terms, an *Evol* role per component. As soon as, via *McPal*, the new way of working together with migration towards it, have been developed and have been installed as an extension of the original model, *McPal* starts coordinating the migration. Its own migration begins, the migration of the others is started thereafter. Finishing migration is done in reversed order. The others are explicitly left to their new stable collaboration phase before *McPal* ceases to influence the others. As a last step, *McPal* shrinks the recently extended model, by removing model fragments no longer needed, keeping the new model only.

It is stressed, migration is on-the-fly. New behaviour per component just emerges in the ongoing execution. Note that no quiescence of components is needed. Additionally, *McPal*'s way of working is pattern-like, as *McPal* can be reused afterward for yet another unforeseen migration.
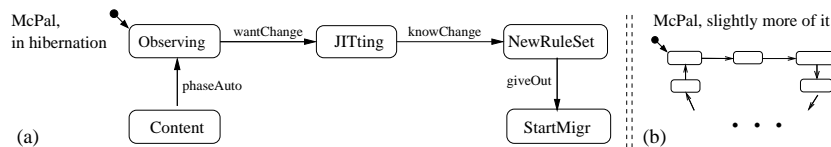

Fig. 2. *McPal*, its hibernated form.

Figure 2(a) visualizes *McPal*'s detailed dynamics in its hibernated form only. In starting state *Observing*, *McPal* is doing nothing in particular, but it can observe, that something should change. State *JITting* is where just-in-time foreseeing and modeling of such a concrete change occurs. The extended model then is available in state *NewRuleSet*. Thus, upon leaving *NewRuleSet* for state *StartMigr*, *McPal* extends its hibernated form with originally unknown dynamics for coordinating the migration. Such an extension is suggested in Figure 2(b).

Figure 3(a) visualizes the ingredients for *McPal*'s role *Evol*. *McPal*'s hibernated form returns here as the phase *Stat*. The other phase *Migr* represents *McPal*'s coordinating a once-only migration. Figure 3(b) visualizes the role STD *McPal(Evol)*. It says, *McPal*'s hibernation constraint is replaced by the migration constraint, after entering trap *ready* (i.e. once state *NewRuleSet* has been reached). Note, the originally unforeseen migration dynamics are known

by then indeed. Similarly, the hibernation constraint is being re-installed after trap *migrDone* has been entered. So, by returning to starting state *Observing* all model fragments obsolete by then, can be safely removed, including the phase *Migr* of *McPal*. Then, the new stable model is in execution, with *McPal* present in its original, hibernated form.
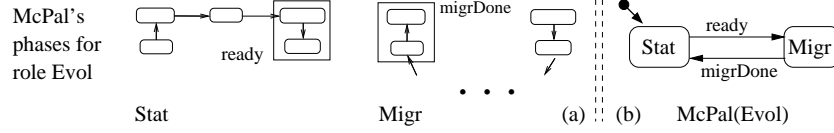

Fig. 3. *McPal*, its phases and global process.

In the style of a horizontal UML activity diagram, Figure 4(a) gives a small part of the coupling between *McPal* and *McPal(Evol)*. Regarding *McPal*, the Paradigm model has initially the following two consistency rules, specifying *McPal*'s first two steps only, the first one without any further coupling.

$McPal$: $Observing \stackrel{wantChange}{\rightarrow} JITting$

$McPal$: $JITting \stackrel{knowChange}{\rightarrow} NewRuleSet * McPal[\, Crs := Crs + Crs_{migr} + Crs_{toBe}\,]$

In the second step from *JITting* to *NewRuleSet*, via a so-called change clause, the set of consistency rules *Crs* for the original stable collaboration is extended with the rules Crs_{migr} for the migration and with the rules Crs_{toBe} for the new, stable collaboration to migrate to. In particular, apart from all other migration coordination details, *McPal* obtains two new consistency rules:

$McPal$: $NewRuleSet \stackrel{giveOut}{\rightarrow} StartMigr * McPal(Evol)$: $Stat \stackrel{ready}{\rightarrow} Migr$

$McPal$: $Content \stackrel{phaseAuto}{\rightarrow} Observing *$
$\qquad McPal(Evol)$: $Migr \stackrel{migrDone}{\rightarrow} Stat, \; McPal[\, Crs := Crs_{toBe}\,]$

The first rule says, on the basis of having entered trap *ready*, the phase change from *Stat* to *Migr* can be made, coupled to *McPal*'s transition from state *NewRuleSet* to *StartMigr*. Figure 4(a) expresses this through the left 'lightning' step. As the last migration step, after having phased out dynamics no longer needed for the other components and eventually having entered trap *migrDone* of its phase *Migr*, *McPal* makes its role *McPal(Evol)* return from *Migr* to *Stat* by making the (coupled) step from state *Content* to *Observing*. Then, also the rule set *Crs* is reduced to Crs_{toBe}, by means of a change clause. See the right 'lightning' in Figure 4(a). Once returned in state *Observing*, *McPal* is in hibernation again, ready for a next migration.

Figure 4(b) suggests how *McPal*, by doing steps between state *StartMigr* and *Content*, may guide other components. Here, one *MU* component migrates from its complete, old dynamics Ph_1 to originally unforeseen dynamics Ph_2, via two intermediate phases $Migr_1$ and $Migr_2$. First, old dynamics is interrupted at trap *triv*. Second, the dynamics is extended after trap *onItsWay* has been entered. Third, finally, the extended dynamics is restricted to that of Ph_2, after trap *ready* has been entered. All this occurs during *McPal*'s migration phase *Migr*.
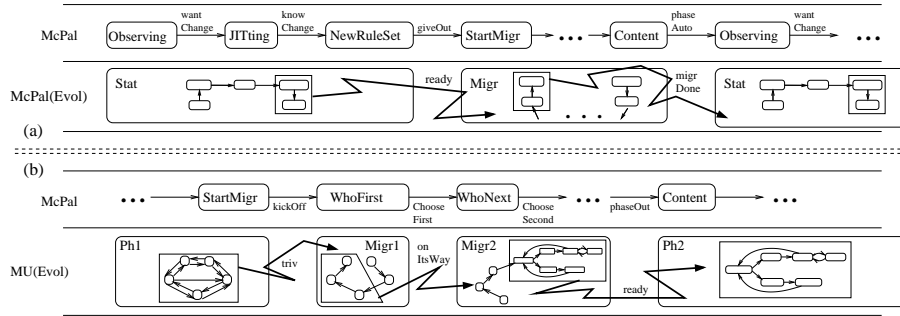
3

Fig. 4. Migration coordination as constraint manipulation.

3 Conclusion

We have sketched how system evolution can be modeled in Paradigm using the migration pattern of *McPal*. New intermediate migration behaviour as well as new target behaviour is added to the relevant components. By restricting the original way of working, components are steered by *McPal* towards a final, stable stage of execution. After removing obsolete model fragments, *McPal* returns to its so-called hibernated form, waiting for a new migration to coordinate.

Paradigm helps structuring software architectures, high-lighting the collaborations that are relevant for separate issues. A prototype environment is reported in [6]. Recently, in [1], a translation of Paradigm into the process algebra ACP is described. This paves the way to state-of-the-art modelchecking using the mCRL2 toolkit [7] developed in Eindhoven, providing support for the verification of invariants and progress properties in Paradigm. Future work is devoted to quantitative analysis of migration, in particular timeliness and performance, envisioning a novel perspective on system migration and evolution. In addition, Paradigm's concept of JIT modeling facilitates that performance triggers *McPal* to update, on-the-fly, the current constraints. Note, Paradigm's constraint handling can be expressed in other languages too, e.g., the UML and ArchiMate.

References

1. S. Andova, L.P.J. Groenewegen, and E.P. de Vink. Dynamic consistency in process algebra: From Paradigm to ACP. In *Proc. FOCLASA'08*. ENTCS, to appear. 19pp.
2. L. Groenewegen, N. van Kampenhout, and E. de Vink. Delegation modeling with Paradigm. In *Proc. Coordination 2005*, pages 94–108. LNCS 3454, 2005.
3. L. Groenewegen and E. de Vink. Operational semantics for coordination in Paradigm. In *Proc. Coordination 2002*, pages 191–206. LNCS 2315, 2002.
4. L. Groenewegen and E. de Vink. Evolution-on-the-fly with Paradigm. In *Proc. Coordination 2006*, pages 97–112. LNCS 4038, 2006.
5. L.P.J. Groenewegen and E.P. de Vink. Dynamic system adaptation by constraint orchestration. Technical Report CSR 08/29, TU/e, 2008. CoRR abs/0811.3492.
6. A.W. Stam. *ParADE – a Conceptual Framework for Software Component Interaction*. PhD thesis, LIACS, Leiden University, 2009. Forthcoming.
7. http://www.mcrl2.org.